1ºProjeto de Lingua Natural

Grupo 26

Cassandro Martinho Nº87642

Rui Nóbrega Nº87703

1. Introduction

For the first project we were assigned to build a retrieval-based chatbot with FAQs from "Balcão do Empreendedor". These FAQs constitute the chatbot's knowledge base (KB.xml), which was built by the different groups for HW2.

The main challenge for this project was to come up with the best way to associate an user request with a question already present on the KB file. For the specific example of: "Demora quanto tempo para emitir um certificado de admissibilidade?" the output of our project should be the id of the answer matching a similar question.

As you will see ahead, our group used the different similarity measures (and edit distances) learnt in the classes such as TF-Idf, Jaccard and edit distance (MED).

2. Proposal

2.1 Pre-processing

As we already know, computers only understand numbers so different variations of a word are interpreted by the machine as totally different. There were a few words in this project that if they were not prepared and handled before using them in the similarity functions we developed, there would be a big contrast with the results we had and the results we would have had. Examples of such words could be:

- "Demora" ≠ "demora"
- "Imitar " ≠ "Imitado"

Although, these words mean or refer to the same thing in a sentence, without a pre-processing they would be as different as "john" and "car" for certain similarity measures.

In order to remove this problem, we used the lowercasing method so all letters of each word could be interpreted as equal. With this "Demora" , for example, no longer was different from "demora".

With the objective of removing the problem of having different conjugations of a verb for example, we resorted to the stemming method. With this, we were able to reduce a word to a more general one.

Not only we used these “tricks” but we also implemented another pre-processing methods such as:

- Elimination of stop words- remove some words that are not very informative for the task in hands, so we ended up with, only, the key words for each user request or question already present in the KB file. The presence of such unnecessary words only represented more calculations and iterations for our functions so their removal would only present better results in terms of efficiency and accuracy due to the fact that when comparing these specific word, if they were present they would match other similar ones.

- Remove punctuation - we decided to remove it due to the fact that not only it was needless but for example the symbol ‘.’ could represent a major problem because it has more than one meaning in a sentence.

2.2 Similarity measures

As previously mention, with the goal of finding the matching question located on the Kb for the user request we used 3 types of similarity measures as you can see implemented on the “Similarity_functions.py”. To note that, each of these algorithms were coded aiming for different results between them so as it should be clear, when one is running the others are not.

The algorithms used, were edit distance, Jaccard and TF-IDF (“Term Frequency — Inverse Document Frequency”).

Different results were obtain with the TF-IDF being the one with the most accurate results. That is because it is a technique used to quantify a word in documents. After running it we get the weight of each word representing the importance in the corpus and in each documents.

The one that didn´t promised a lot was the MED. The MED takes two words at a time and compares them so, as we were expecting, it took a lot of time to run this particular algorithm for the entire test set.

3. Corpora

The corpora used was the same as in the second homework. As requested in that assignment, based in one simple question, we created more examples of that matter. We basically asked the same but in other words.

4. Experimental results

In order to evaluate our different setups, we developed by hand what we believed to be the answers to the test.txt file given. With this in mind, we can now compare our answer to the output of our program and calculate an estimation of their accuracy. Note that 50 examples are not a lot and so, our accuracy is an estimation. Furthermore, our test set doesn't vary much, that is, some entries are very similar and so, it could influence the accuracy.

4.1 MED

Setup nº1 (accuracy = 35%): We use all the pre-processing techniques defined previously and an Edit Distance similarity metric. This setup ended up being very slow (8 minutes) which was expected due to the high complexity of comparing every question in the corpora with the objective.

4.2 Jaccard

Setup nº2 (accuracy = 68%): We used all the pre-processing techniques defined previously and a Jaccard similarity metric. This setup turned out to be average which surprised us a little. We had to setup a threshold (0.75) for the Jaccard value in order to determine what questions did not fit any of the questions in the corpora.

Setup nº3 (accuracy = 76%): We used the Jaccard similarity metric and all pre-processing defined previously except the stemming method. This was a curious outcome as the use of stemming seemed to lower the accuracy of the Jaccard method. With this change words like "comprovo" and "comprovar" will not be viewed as the same word by the Jaccard method, which at first, we thought would just lower the accuracy even further. Of course, it could be an error since we have such a small test set.

4.3 TF-IDF

The way we developed Tf-idf was so every set of questions (group of questions asking the same thing) is defined as a Document. So, in the end we have a list of 615 documents.

Setup nº4 (accuracy = 88%): We used all the pre-processing techniques defined previously and a Tf-idf similarity metric. We were very happy to see that this was the best approach since it makes sense to be the best way to address our problem since it can understand witch words are important in a user input and choose an existing document that is like it.

Setup nº5 (accuracy = 84%): We used a Tf-idf similarity metric and all the pre-processing techniques defined previously except the stemming method. We were very curious to see if the accuracy would rise without the stemming method just as it happens with the Jaccard method. As we thought, the stemming helps the Tf-idf method because it makes the counting of each word a way much simpler task. For example, the words "submeti" and "submeter" would count as an appearance of the same word and not two different words.

5. Conclusions and Future work

In conclusion, our program has an 88% chance of answering a user question correctly using a Tf-idf method and a few pre-processing techniques.

For future work, we could consider N-gram, but depending on implementation it could be very memory heavy. Anything bigger than bigrams can require very big arrays. We also need to pay attention to the need of some smoothing techniques in order to predict better values.

6. Bibliography

For this project were used the following libraries:

- NLTK
- Pandas
- Sklearn
- Unidecode
- xml.etree.ElementTree
- timeit
- sys